IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF VIRGINIA
RICHMOND DIVISION

THE TRUSTEES OF COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK, *Plaintiff* vs. SYMANTEC CORPORATION, *Defendant*	Civil Action No. 3:13-cv-00808-JRS **JURY TRIAL DEMANDED**

<u>DECLARATION OF PROFESSOR DOUGLAS C. SZAJDA</u>

TABLE OF CONTENTS

Page

I. Background and Qualifications

1. I am a professor with tenure in the Department of Mathematics and Computer Science at the University of Richmond. I received my PhD in Mathematics and a Masters of Computer Science from the University of Virginia in 1999. I then held a post-doctoral fellowship in Computer Science at the University of Maryland Institute for Advanced Computer Studies. Exhibit A is a copy of my CV. All exhibits in my declaration are in the Declaration of Gavin Snyder ("Snyder Decl.").

2. Several aspects of my professional life are relevant to the subject of this declaration. First, I train computer scientists in aspects of computer security directly relevant to the three families of Columbia patents that I understand are issue in this case. I teach Computer Networks and Computer Security classes in my department. In addition, under the auspices of programs such as the National Science Foundation Cyber Trust program grant, I train computer security researchers in the laboratory. These students conduct research at the top universities and technology companies in the country, including Microsoft and Google. I also have been the coordinator of the University of Richmond's System Security Group since 2002.

3. Second, outside of the University I have devoted a large portion of my professional life to issues of computer security. I served as General Chair of the Internet Society's Network and Distributed System Security ("NDSS") Symposium from 2008–2011, as an NDSS steering group member since 2007, and as a member of the conference organizing committee from 2005–2007. I have served on program committees for NDSS and the security track of the International Conference on Security Data Services. I have also reviewed papers for both the IEEE Symposium on Security and Privacy and the USENIX Security Symposium. These are some of the most prominent conferences on computer security in the world.

4. Third, the research group I lead at the University of Richmond is focused on applying the same type of technology described in the three patent families at issue in this case: using machine learning techniques based on artificial intelligence to detect whether web pages contain malicious programs (*e.g.*, via embedded scripts or through links that cause malicious scripts to be downloaded and executed). Indeed, we have a constructed a working platform that can perform real-time analysis of web pages to detect if they are hosting malicious programs. The platform has three parts: an instrumented web crawler for collecting candidate pages, an extraction unit to extract relevant features of the pages, and an analysis unit, which creates artificial intelligence models. The prototype is capable of mining virtually any data that is freely available over the Internet, and, with slight modification, can potentially perform analysis of any network transported malware.

II. Legal Standards Applied

5. Appendix A lists the legal standards I have been asked to apply in my analysis and discussion.

III. Subject of the Declaration and Basis for Opinions

6. I have been asked to provide background information on the technology in the three families of Columbia patents at issue in the case. As part of this process, I have also provided a summary of how a person of ordinary skill in the art of the patents would understand a number of the concepts that I understand are at issue in the proceedings. In preparing this declaration I have relied on my extensive experience in the field, as well the materials referenced in this declaration and certain material listed in Appendix B.

IV. Background on Computer Security and Anti-Virus Technology

7. Computing devices have steadily increased in utility and prevalence since their introduction in the mid-20th century. The first computers were expensive, large mainframes. Personal computers were introduced in the 1970s and 1980s. Today, the frontier of computing technologies is smartphones and tablets augmented by computing done at remote locations on the Internet (cloud computing). Wearable devices, ubiquitous computing in everyday objects, and robotics are on the near horizon. It is not an exaggeration to say that the use of computing devices has become a hallmark of our modern lives.

8. However, since personal computers became widespread, a major issue with the safety and security of computing devices has been intentionally malicious programs, also known by various other names such as "viruses" or "malware." These programs can do many different undesirable things, including damaging the devices they run on, causing programs to crash or not run properly, causing data to be lost, capturing confidential personal information (such as passwords or financial records), or enabling the device to be controlled by a remote "botmaster," which is to say, a remote entity. These are just some examples.

9. The growth of the Internet has created additional challenges by providing a platform for malicious programs to take on additional strengths and capabilities. For example, if a malicious program takes over a device, it can duplicate itself across a network, and introduce the infection to even more devices. The growth of email communication further compounds the problem. In particular, a malicious program could be delivered as an executable email attachment. A user might receive a benign-looking email, such as an email appearing to be from a friend, with a malicious email attachment. When the user opens the attachment, the attachment attacks the user's computer and compromises its security. Then, the malicious program

replicates its code and sends out even more malicious emails with itself as an email attachment, perhaps to everyone in the user's address book. Email attachments present an especially dangerous method of malicious program distribution because of the ease with which malware can be distributed, at low cost, to millions of potential victims.

10. The economic loss from malicious programs can be enormous, and the threat increases every year. In 2000, the ILOVEYOU virus executable directed the victim's computer to send to his or her entire contact list the same executable that infected the original user's computer. In this way, the virus spread to over 420,000 Internet hosts during the first day it was reported. More recently, Ukrainian hackers used a malicious program to infiltrate the retailer Target's point-of-sale terminals in 2013 and were able to steal over 40 million credit card numbers. *See* Ex. B to Snyder Decl., Michael Riley, Ben Elgin, Dune Lawrence & Carol Matlack, *Missed Alarms and 40 Million Stolen Credit Card Numbers: How Target Blew It*, Bloomberg Businessweek (Mar. 13, 2014), http://www.businessweek.com/articles/2014-03-13/target-missed-alarms-in-epic-hack-of-credit-card-data.

11. The potential (and actual) damage from malware has fueled a growing security and anti-virus industry, as well as extensive academic research, often supported by government funding, into ways to protect computing devices. Numerous security companies offer many different types of security products to protect end user computers, servers (such as those which manage email), and other network equipment.

12. One of the major problems in computer security is how to distinguish malicious programs from benign ones. To the operating system, a malicious program is not intrinsically different from a benign program. If a program replicates itself without changing the host device in a way that would be apparent to the user, the user may not know that his or her device has

been compromised. If malicious programs cannot be distinguished from benign programs, then malicious programs cannot be stopped or quarantined.

13. One method for detecting malicious programs is the use of signatures. Typically, an anti-virus analyst makes a manual identification of a malicious program. A signature identifying the virus is then created and that signature is used to detect copies of the malicious program should it appear. Typically, the signature is compared to every new file on a computing device. If the signature matches the file, the file is flagged as malicious.

14. Signatures have various shortcomings. For example, if a variant of a particular virus is developed, the old signature cannot detect the new variant. A new signature needs to be developed. In addition to being unable to detect variants of existing viruses, signatures cannot detect new threats, often called "zero-day[1] attacks," that have never been seen before—because no signature existed for the zero-day attack. Another problem with signatures is apparent when one considers the rate at which new malicious programs are created and disseminated. The number of new malicious programs created every day has been steadily increasing. Because signatures typically rely on a human identifying a piece of code as malicious before a signature is created, the number of new viruses could overwhelm a limited staff of anti-virus analysts.

15. The patents at issue in this case concern a newer method of detecting a malicious program, different from signatures. This approach uses a technique called machine learning. Machine learning is a sub-field of computer science and artificial intelligence in which systems are constructed to learn from data. The first step in the machine learning process is to collect a

[1] The attacks are referred to as "zero day" because there are zero days to prepare for the attacks and develop a patch that would remove the vulnerability.

large amount of data. From there, a machine learning algorithm[2] can be applied to the data to recognize patterns or distinguish between different types (or "classes") of data in the collection. The system "learns" rules that it can apply to new inputs. The rules are combined and encapsulated in machine learning "models." The process of providing the system with data and the subsequent learning is referred to as "training." After a machine learning model has been trained, new input can be compared to the model. The system applies the rules in the model to the new input and renders a verdict on the input. For example, the system could predict that the new input is a member of a particular class of data that was observed in the training stage.

16. Consider a machine learning system for apples. The system starts with a set of training data: a basket of apples and oranges. By studying the example fruit in the basket, the system learns rules to effectively distinguish between apples and oranges. For example, one rule could be that oranges have orange skin. Another rule could be that apples have hard cores. With these rules in place, when the machine learning system encounters a new piece of fruit, the system can determine that the fruit is an apple.

17. The patents at issue in this case relate to the work of Professor Salvatore Stolfo and others in his Intrusion Detection Systems ("IDS") Lab at Columbia University, as well as the work of Professor Angelos Keromytis and his students. Professors Stolfo and Keromytis are well-known in the field. I have reviewed a number of the publications from their laboratories. At the turn of the century, Professor Stolfo was applying machine learning based on artificial intelligence to computer security and malware detection. He assembled a team of graduate and

[2] For our purposes, an algorithm is a step by step procedure for data processing performed by a computer.

undergraduate students to work in the IDS Lab on a comprehensive suite of security strategies involving the use of machine learning based on artificial intelligence techniques.

18. Professor Stolfo's research teams then performed the work that resulted in the patents at issue in the case. As discussed in the relevant patents, one team worked on a system to detect malicious executable email attachments using a model derived from an analysis of programs. Using the model, the system could detect with high accuracy whether new programs in email attachments were malicious or not. This work served as a basis for United States Patents 7,487,544 and 7,979,907. Ex. C to Snyder Decl. ('544 patent); Ex. D to Snyder Decl. ('907 patent).

19. As discussed in the relevant patents, another team worked on understanding patterns of access to the operating system registry. The team used machine learning techniques to develop models of registry activity. The operating system registry is utilized by Windows. It consists of a hierarchical database used to store configuration information, in the form of keys and values. Programs running on Windows systems can use the registry to store their configuration information and other information necessary for the program to run. But programs may also access portions of the registry that belong to the operating system or to other programs. The team realized that patterns of registry access could indicate whether the program accessing the registry was acting in an anomalous manner, and therefore potentially malicious. The team also realized that the techniques that worked with the Windows registry could be extended to the file system of a computer. This was a basis for United States Patents 7,448,084 and 7,913,306. Ex. E to Snyder Decl. ('084 patent); Ex. F to Snyder Decl. ('306 patent).

20. The last family of patents at issue in this case, United States Patents 8,074,115 and 8,601,322, represents a combination of machine learning based on artificial intelligence

technology and emulation technology that could monitor and selectively execute all or a part of a program. Ex. G to Snyder Decl. ('115 patent); Ex. H to Snyder Decl. ('322 patent). The combination of these technologies could be used to monitor a program and compare its activity to a model of activity to determine if there was an anomaly indicative of a malicious program or an attack. If an anomaly was detected, other members of an application community could be notified by the system.

V. Person Ordinary Skill in the Art

21. I understand that claim interpretation is from the perspective of a person of ordinary skill in the art at the time of the invention. In my opinion a person of ordinary skill in the art in the three patent families would have an undergraduate degree in computer science or mathematics, and one to two years of experience in the field of computer security.[3] I believe this is an appropriate definition of ordinary skill for two reasons. First, in the academic environment, a person of this level of experience could be beginning to enter the research phase of their PhD. At this point they would be beginning to design security systems that they would then study and analyze. Second, in the commercial environment, a person with this level of experience would be part of the group making design decisions for code in security systems.

22. Other facts that influence my opinion include the following: although the technology at issue is complex and evolves rapidly, there are instances in which unique workers who lack a college degree have made important contributions in the field. For example, of the inventors in the three families of patents, I understand that the following were undergraduate

[3] For purposes of construing the meaning of the claims to a person of skill in the art, no opinion I give changes if a higher or slightly lower level of skill is assumed. This is because the concepts discussed in the patent do not have different meanings depending on the amount of experience (above a minimum) that one possesses in this space.

students during at least some of their work in the IDS lab: Matthew Schultz, Manasi Bhattacharyya, Frank Apap, and Andrew Honig. In addition, I understand the following were PhD students: Eleazar Eskin, Erez Zadok, Shlomo Hershkop, and Stelios Sidiroglou-Douskos. Placing the level of ordinary skill too high would exclude some of the very individuals who invented the technology we are discussing.

23. All of my opinions about a patent family are given from the standpoint of a person of ordinary skill in the art as the of the invention date of the patent family.

VI. ’544/’907 Patents

24. United States Patent Nos. 7,487,544 and 7,979,907 share a common parent application filed on July 30, 2002, based on a provisional application filed on July 30, 2001. I understand that I am to provide my opinions on the understanding of concepts by a person of ordinary skill as of the filing date of the application. No opinion I give changes if the date selected is July 2002 or July 2001.

B. Background on the Patents

25. To understand the ’544/’907 patents, it is important to have some context on the issues the inventors were facing when they did their work. The most traditional approach to preventing malicious email attachments is a signature-based approach. In a signature-based approach, the distributor of anti-virus software would first identify malicious executables, and then “create a unique tag for each malicious program” encountered. Ex. C to Snyder Decl., ’544 patent at 1:54–57. Then, when that exact tag is encountered in the future, the anti-virus software can accurately identify the file as malicious. Because even a slight change in an executable can potentially evade detection, signature-based methods “do not generalize well to detect new malicious binaries [or programs].” *Id.* at 1:57–59.

26. Other approaches had also been attempted, but were either highly inefficient or only applicable to certain types of executables. *Id.* at 1:63–66, 2:26–29.

27. The inventors recognized these shortcomings of the prior art, and recognized that a security system could be trained using data mining techniques to better recognize and prevent zero-day attacks (attacks by programs that have not been encountered before). *Id*. at 1:34–2:65 (Background of the Invention).

28. One example of how the inventors solved these shortcomings is disclosed in the claim 1 of the '544 patent:

> 1. A method for classifying an executable attachment in an email received at an email processing application of a computer system comprising:
>
>   a. filtering said executable attachment from said email;
>
>   b. extracting a byte sequence feature from said executable attachment; and
>
>   c. classifying said executable attachment by comparing said byte sequence feature of said executable attachment with a classification rule set derived from byte sequence features of a set of executables having a predetermined class in a set of classes to determine the probability whether said executable attachment is malicious, wherein extracting said byte sequence features from said executable attachment comprises creating a byte string representative of resources referenced by said executable attachment.

29. In this claim, a byte sequence feature is extracted from a potentially suspicious executable attached to an email. This byte sequence feature comprises a specific type of information: "a byte string representative of resources referenced by said executable attachment." This concept will be discussed in detail below.

30. After a byte sequence feature is extracted, it is compared to a model created utilizing machine learning techniques: "a classification rule set derived from byte sequence features of a set of executables having a predetermined class in a set of classes." From the comparison, the model predicts whether the executable is malicious.

31. Although I cannot address each embodiment or technique claimed by the ’544/’907 patents in this short declaration, I can discuss some fundamental concepts that are most relevant to the issues presented to the Court in the claim construction process.

C. Creation of the Byte Sequence Feature

32. As noted above, claims of the ’544/’907 patents require extracting a byte sequence feature from an executable. *E.g.*, Ex. C to Snyder Decl.,’544 claims 1, 28; Ex. D to Snyder Decl., ’907 claims 1, 10. Some claims of the ’544/’907 patents (*e.g.*, ’544 patent claim 1, above) contain the further limitation: “wherein the byte sequence features include a byte string representative of resources referenced by the executable attachment.”

33. First, I will give an example of how a person of ordinary skill in the art would understand this claim language works in practice and then, second, I will discuss the claim language generally from the standpoint of a person of ordinary skill in the art.

34. The patent gives as an example of resources from which a byte string can be generated as “[t]he list of DLLs used by the binary.” Ex. C to Snyder Decl., ’544 patent at 6:59–61. A DLL (“Dynamic Link Library”) is a shared library of information that can be accessed by many different programs on a computer. Different DLLs allow their calling programs to do different things—for example, to send a network packet or perform advanced math. As a result, understanding what DLLs an executable uses can reveal how the executable behaves. In one example in the specification, an executable accesses GDI32.DLL, a DLL that relates to the creation of two-dimensional information on the computer screen, and WINNM.DLL, which relates to the user interface process. *Id.* at 7:4–11. These are examples of “resources” accessed by the executable. A string of bytes is generated which represents the fact that the executable accesses these specific resources. This is an example of information that can be present in byte sequence features extracted from an executable.

35. With this in mind, I can address some of the subsidiary concepts in the claim:

ii. Bits and Bytes

36. In computer science, all data can be reduced to ones and zeros. Each one or zero is called a "bit," which is short for a "binary digit." *See* Ex. I to Snyder Decl., McGraw-Hill Dictionary of Scientific and Technical Terms 246 (6th ed. 2002) ("bit [COMPUT SCI] 1. A unit of information content equal to one binary decision or the designation of one of two possible or equally likely values or states of anything used to store or convey information.").

37. In order to represent larger amounts of information, bits are generally grouped together. A grouping of eight bits is traditionally viewed as the classic example of a byte. *See id.* 305 ("byte [COMPUT SCI] A sequence of adjacent binary digits operated upon as a unit in a computer and usually shorter than a word.").

iii. Byte Sequence

38. As noted above, a "byte" is group of eight bits. And a "byte sequence" is simply a sequence of bytes.

iv. Byte Sequence Feature

39. The patent explains that "a feature is a property or attribute of data (such as a 'byte sequence feature') which may take on a set of values." Ex. C to Snyder Decl., '544 patent at 5:61–64.

40. Taken together, a "byte sequence feature" is a feature or property of a sequence of bytes, which may take on a set of values. Some claims, such as claim 1 of the '544 patent, also contain a subsequent limitation that specifies the information that the byte sequence feature must represent.

v. Extracting a Byte Sequence Feature by Creating a Byte String Representative of Resources

41. Claim 1 of the ’544 patent recites “extracting a byte sequence feature from said executable attachment . . . wherein extracting said byte sequence features from said executable attachment comprises creating a byte string representative of resources referenced by said executable attachment.” *See id.* claim 1. Thus, the claimed byte sequence features will comprise “a byte string representative of resources.” This is a clear concept to a person of ordinary skill: The “a byte sequence feature” of claim 1 must include a very particular class of information: a string of bytes representative of resources referenced by the executable. The claimed “a byte sequence feature” can have other things, but it must include data on resources referenced by the executable.

vi. Instructions Executed by the Central Processing Unit Are Not the Only Source for Byte Sequence Features

42. Executable programs can contain a wide array of information. One piece of information is the instructions that are performed by a computer’s central processing unit (“CPU”). But CPU instructions are not the only thing in an executable. They are also not the only type of byte sequence features. For example, the patent teaches creating byte sequence features based on the list of DLLs used by the program, the number of different functions in those DLLS that are called by the program, and information written in plain text in portions of the program. *See, e.g.*, *id.* at 6:64–7:12, 7:43–45 (“Headers in PE format are in plain text, which allows extraction of the same information from the PE executables by extracting the plain text header.”). None of these would be classically described as instructions performed by the central processing unit, but all are recognized as part of an executable. *See* Ex. J to Snyder Decl., Lee Holmes, *Windows PowerShell Cookbook* (2010) (noting that “The beginning section of any executable (a .DLL, .EXE, or any of several others) starts with a binary section known as the

portable executable (PE) header. Part of this header includes characteristics about that file, such as whether the file is a DLL."); *see also* Ex. K to Snyder Decl., Microsoft Corporation, *Microsoft Portable Executable and Common Object File Format Specification Revision 6.0* (1999) (discussing different parts of executable).

vii. The Use of Hexdump to Extract a Byte Sequence Feature Is Not Required

43. In describing particular exemplary embodiments, the specification discloses the use of a program known as "hexdump" as one manner of extracting a byte sequence feature. Ex. C to Snyder Decl., '544 patent at 6:7–22 ("In the exemplary embodiment, hexdump was used in the feature extraction step."). As described by the specification and known in the art, the hexdump program "is an open source tool [i.e., a program] that transforms binary files into hexadecimal[4] files." *Id*. at 6:11–12. In the exemplary embodiment involving hexdump, "[e]ach byte sequence" from the file "is used as a feature." *Id*. at 6:21–22. In other words, hexdump creates byte strings for ***all*** data in the executable. This includes not just machine code instructions that are executed, but plain text headers, library information, and anything else in the program. In order to verify this fact, I obtained the hexdump program and confirmed that it converted all the data in the program, not just instructions, into hexadecimal format.

44. Hexdump generates its byte sequence features in hexadecimal format. There are a number of claims in the '544/'907 patents that expressly refer to hexadecimal format for the byte sequence feature. *See*, *e.g*., *id.*, claims 3, 16, 30, 43; Ex. D to Snyder Decl., '907 patent, claims 3, 12. Limitations specifically requiring a hexadecimal format, however, do not appear in claim

[4] Hexadecimal formatting converts a byte (8 bits of data) into a number consisting of two hexadecimal (base 16) digits.

1 of the ’544 patent or in many other independent claims in the patents. *See* Ex. C to Snyder Decl., ’544 patent, claims 1, 6, 34; Ex. D to Snyder Decl., ’907 patent, claims 1, 10.

D. Email Interface

45. The specification speaks about the use of an “email interface.” An interface was a common structure in computer science at the time of the patents: a point of interaction between two things. This meaning has remained steady since the date of the patents and remains the same today. *See* Ex. I to Snyder Decl., McGraw-Hill Dictionary of Scientific and Technical Terms 1093 (6th ed. 2002) (“interface [COMPUTER SCI] 1. Some form of electronic device that enables one piece of gear to communicate with or control another.”). The usage in the patents is consistent with this common understanding of the concept.

46. The specification explains that an email interface may optionally perform a number of different particular tasks. For example, after an executable is analyzed, the email interface can:

- “reintegrate filtered emails back into normal traffic”
- “send the model generator . . . each attachment to be further analyzed.”
- “add warnings to the email”
- “quarantine the email”
- “log” the executable;
- “increment a count” of executables that are “classified as borderline”
- “provide a notification” that a threshold has been exceeded.

Ex. C to Snyder Decl., ’544 patent at 15:30–37; claims.32, 41–42. The functions are distinct options that are not necessarily dependent on each other. In other words, for example, you do not need to be able to “reintegrate filtered emails” in order to be able to “send [the attachment to]

the model generator," "log" the executable," "increment a count," or "provide a notification." The listing shows the great diversity of functions that can be performed by an email interface. But what the functions all have in common is that the interface has the ability to interact with emails as well as the structures that are used in the patents to analyze the emails.

VII. '084/'306 Patents

47. United States Patent Nos. 7,448,084 and 7,913,306 share a common parent application filed on January 27, 2003, based on a provisional application filed on January 25, 2002. I understand that I am to provide my opinions on the understanding of concepts by a person of ordinary skill as of the filing date of the application. My opinions set forth in this declaration are the same whether the date selected is January 2003 or January 2002.

A. Background on the '084/'306 Patents

48. The '084/'306 patents are about detecting the presence of a malicious program on a computer system by identifying abnormal behavior affecting the operating system registry or the file system. An operating system registry is a unique feature of Windows. Because Microsoft Windows is so widely used, systems running Windows are frequently subject to malware attacks.

49. The '084/'306 patents are set against the backdrop of the approach of detecting malicious software attacks by scanning for known viruses. Virus scanners at the time of the patents were traditionally "signature-based, which generally means that they use byte sequences or embedded strings in software to identify certain programs as malicious." Ex. E to Snyder Decl., '084 patent at 1:63–65. The virus scanner software provider typically created a database containing the signatures for the malicious programs that had been detected previously. "If a virus scanner's signature database does not contain a signature for a malicious program, the virus scanner is unable to detect or protect against that malicious program." *Id.* at 1:65–2:1.

Generally, strategies that look *only* at known malicious data are unable to anticipate and detect new viruses as they appear.

50. The inventors on the ’084/’306 patents recognized that a more effective way to detect intrusions from new and unseen attacks could involve what they referred to as anomaly detection using models of normal behavior. “Anomaly detection algorithms may build models of normal behavior in order to detect behavior that deviates from normal behavior and which may correspond to an attack.” *Id.* at 2:34–37. In other words, understanding how a normal system acts differently from an infected system can facilitate the determination of whether an unknown program will cause a system to operate in a manner that deviates from normal.

51. Claim 14 of the ’084 patent is exemplary:

> 14. A system for detecting intrusions in the operation of a computer system comprising:
>
> (a) an operating system registry;
>
> (b) a registry auditing module configured to gather records regarding processes that access the operating system registry;
>
> (c) a model generator configured to generate a probabilistic model of normal computer system usage based on records of a plurality of processes that access the operating system registry and that are indicative of normal computer system usage and to determine the likelihood of observing a process that was not observed in the records of the plurality of processes that access the operating system registry and that are indicative of normal computer usage; and
>
> (d) a model comparator configured to receive the probab1istic model of normal computer system usage and to receive records regarding processes that access the operating system registry and to detect deviations from normal computer system usage to determine whether the access of the operating system registry is an anomaly.

B. Operating System Registry

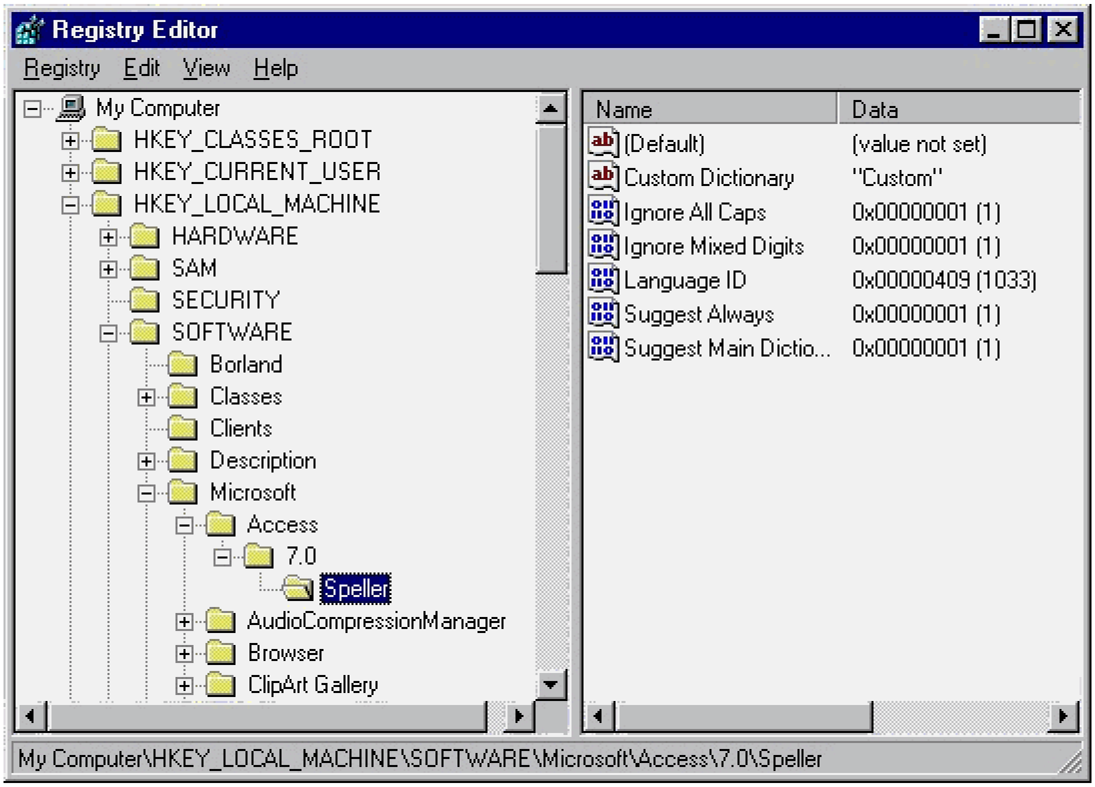
52. An operating system registry is a well-known component of computer systems running Windows. The inventors observed that, among other things, patterns of access to the

operating system registry could be a way to detect malicious software. At the time of the patent Windows was widely used, so concentrating on the registry could detect many threats. Windows remains extremely popular through the present.

53. The registry is a specific database of particular information about a computer's configuration and is organized hierarchically as a tree containing entries consisting of keys and values:

> ***As is known in the art, the registry is a database of information about a computer's configuration***. The registry contains information that is continually referenced by many different programs during the operation of the computer system. The registry may store information concerning the hardware installed on the system, the ports that are being used, profiles for each user, configuration settings for programs, and many other parameters of the system. The registry is the main storage location for all configuration information for almost all programs. The registry also stores much of the important configuration information that are needed by programs in order to run. [¶] ***The registry is organized hierarchically as a tree. Each entry in the registry is called a 'key' and has an associated value.*** *Id.* at 5:20–36 (emphasis added).

The registry contains entries identified by a "key" and an associated "value." *Id.* at 5:36–38. For illustrative purposes, below is a screen shot, from approximately the time the patents were filed, of the contents of a portion of the operating system registry on a computer running Windows NT.



As can be seen from the screenshot, the registry is organized hierarchically as a tree, with nested entries of keys and values. In this screenshot, it can be seen how the registry stores keys (listed as "Name") and corresponding values (listed as "Data") associated with configuration options for the Microsoft Access program. As another example, the specification of the '084 patent describes a registry entry to store a password for AOL Instant Messenger, a popular messaging program at the time the patents were filed. *Id.* at 5:42–60.

54. Because many malicious programs interact with the registry in a manner that is harmful or disruptive, the inventors realized that auditing the registry would provide a useful data source that could be used to detect intrusions. *Id.* at 5:61–6:8. The inventors also noted that "[s]everal advantages of monitoring the registry include the fact that registry activity is regular by nature, that the registry can be monitored with low computational overhead, and that almost all system activities query the registry." *Id.* at 5:6–9.

55. The operating system registry is a database with a unique structure and contents. But the inventors recognized that they could also monitor file systems using the techniques originally developed for the registry. Their techniques could thus be applied as well to computer systems that have file systems but lack registries:

> Another exemplary embodiment is described herein. The systems and method described above, uses Windows™ registry accesses, which is an example of a general means of 45 detecting malicious uses of a host computer. ***However, other systems, such as Linux/Unix, do not use a registry. In those cases, a file system sensor would be used***. Accesses to the file system provides an audit source of data (i.e., whenever an application is run, any and all things accessed are files, e.g., the main executable files are accessed, and the project files are accessed). This audit source can be observed, and a "normal baseline model" built, and then used for detecting abnormal file system accesses. *Id.* at 17:43–54 (emphasis added).

The Linux/Unix file systems, which are not operating system registries, contain configuration information. For example, an article by IBM describes the type of configuration files present in Linux. Ex. L to Snyder Decl., Subodh Soni, *Understanding Linux configuration files*, IBM developerWorks (December 1, 2001), http://www.ibm.com/developerworks/library/l-config/. A similar article by Red Hat, one of the leading providers of Linux, provides the same type of information. Ex. M to Snyder Decl., *Configuring OpenSSH*, Red Hat Enterprise Linux, https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/System_Administrators_Guide/s1-ssh-configuration.html.

C. Anomaly

56. The'084/'306 patents are about detecting intrusions into a computer system by determining if activity is anomalous. An anomaly, in the context of computer security, is "behavior that deviates from normal and may correspond to an attack." This is consistent with the specification of the '084 patent: "Anomaly detection algorithms may build models of normal behavior in order to detect behavior that deviates from normal behavior and which may

correspond to an attack." Ex. E to Snyder Decl., '084 patent at 2:34–37; *see also id.* at 3:17–30: ("It is another object of the invention to generate a model of the normal access to the Windows registry, and to detect anomalous accesses to the registry that are ***indicative of attacks***.) (emphasis added); *id.* at 7:48–49 (anomaly detectors "look for ***deviations from normal activity***) (emphasis added). Consequently, such deviations, which represent normal operation, may nevertheless be declared an attack by the system.").

D. Probabilistic Model of Normal Computer System Usage

57. The inventors note at the beginning of the '084/'306 patents, in the "Background of the Invention," that there are different ways to detect intrusions. The inventors first describe signature-based algorithms:

> **[Option 1]** Typically, these algorithms [of commercial intrusion detection systems in use] match host activity to a database of signatures which correspond to known attacks. This approach, like virus detection algorithms, requires previous knowledge of an attack and is rarely effective on new attacks. *Id.* at 2: 28–32.

Then, the inventors describe a form of anomaly detection involving models of normal behavior:

> **[Option 2]** However, recently there has been growing interest in the use of data mining techniques, such as anomaly detection, in IDS systems. Anomaly detection algorithms may build models of normal behavior in order to detect behavior that deviates from normal behavior and which may correspond to an attack. One important advantage of anomaly detection is that it may detect new attacks, and consequently may be an effective defense against new malicious software. Anomaly detection algorithms have been applied to network intrusion detection see, e.g., D. E. Denning, "An Intrusion Detection Model, *IEEE Transactions on Software Engineering*, SE-13: 222–232, 1987; H. S. Javitz and A. Valdes, "The NIDES Statistical Component: Description and Justification, *Technical report, SRI International*, 1993; and W. Lee, S. J. Stolfo, and K. Mok, "Data Mining in Work Flow Environments: Experiences in Intrusion Detection," *Proceedings of the* 1999 *Conference on Knowledge Discovery and Data Mining* (KDD-99), 1999) and also to the analysis of system calls for host based intrusion detection (see, e.g., Stephanie Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A Sense of Selffor UNIX Processes," *IEEE Computer Society*, pp. 120- 128,1996; Christina Warrender, Stephanie Forrest, and Barak Pearlmutter, "Detecting Intrusions Using System Calls: Alternative Data Models," *IEEE Computer Society*, pp. 133–145, 1999; S. A. Hofineyr, Stephanie Forrest, and A.

> Somayaji, "Intrusion Detect Using Sequences of System Calls," *Journal of Computer Security*, 6:151–180, 1998; W. Lee, S. J. Stolfo, and P. K. Chan, "Learning Patterns from UNIX Processes Execution Traces for Intrusion Detection," AAAI Press, pp. 50–56,1997; and Eleazar Eskin, "Anomaly Detection Over Noisy Data Using Learned Probability Distributions," *Proceedings of the Seventeenth International Conference on Machine Learning* (ICML-2000), 2000). Ex. E to Snyder Decl., '084 patent at 2:32–64.

58. Option 1 thus was a database populated solely with "signatures" corresponding to "known attacks." Option 2 was anomaly detection based on a model of "normal behavior in order to detect behavior that deviates from normal behavior and which may correspond to an attack." A model of normal behavior can certainly be built with data on normal activity. But it can include other data, such as data on abnormal activity, as a supplement. In academic research and in industry, both at the time of the patents, and at present, an anomaly detection model of normal computer system usage is often created by considering not only data on how normal systems operate, but also data on how infected systems operate. In other words, the model need not use ***only*** normal data; it can be built with normal data supplemented with abnormal data. Indeed, several articles cited in the patents describe performing anomaly detection based on models with data from both normal and abnormal activity. The articles are from Professor Stolfo's laboratory.

59. I will first describe why enriching a data set with information on abnormal activity can make for a more robust model of normal activity. I will then discuss the teachings of the references listed in the patent specification.

60. The following hypothetical can help to explain the benefit of including data on both abnormal and normal activity in developing a model of normal behavior:

61. Assume I study four normal programs that all perform Function A. This suggests that Function A is normal.

62. If I encounter a new program (the "test program") that performs Function A, I might conclude that this is normal as well.

63. But now imagine that I add in additional data, which shows that malicious programs also perform Function A, but when they perform Function A they always do so in combination with Function 1.

64. This allows me to create a more robust model of normal operation: normal operation involves Function A, as long as it is not in combination with Function 1.

65. At this point, let's go back and examine my test program. The test program performs Function A, but I observe that it also performs Function 1. Based on the above, I can now treat the test program as anomalous.

66. Because I have enriched the data I used for my model of normal operation to include both normal and abnormal examples, I am able to more accurately determine whether a program is or is not anomalous.

67. As noted above, the inventors published a number of interesting papers on this concept of constructing a model of normal behavior by analyzing normal data, but supplementing with abnormal data. For example, a paper by Eleazar Eskin, notes that although "typical" anomaly detection methods "require training over clean data (normal data with no anomalies)," "[t]here are several inherent drawbacks to this approach." Ex. N to Snyder Decl., Eleazar Eskin, *Anomaly Detection over Noisy Data using Learned Probability Distributions*, Proceedings of the International Conference on Machine Learning. (2000) at 1. Because of these drawbacks, Inventor Eskin describes the use of anomaly detection that does not involve clean data (*i.e.*, data that is 100% normal), but instead involves normal data that has mixed within it anomalies: "this paper describes a technique for detecting anomalies without clean data."

Inventor Eskin explains how the use of data that has mixed clean and anomalous data can actually enrich the model created: "the anomalies themselves can be of interest as they may show rarely occurring events." *Id*. In other words, the anomalous data can provide intelligence on how to interpret rarely occurring activity.

68. The data used by Inventor Eskin "contains up to 15 months of normal traces for certain programs [normal data] as well as intrusion traces [abnormal data]. The data provides normal and intrusion traces of system calls for several processes." *Id*. at 4. The study shows that the use of normal data supplemented with abnormal data is more effective than a system that trains with only normal data: "We first empirically show that the method presented in this paper out performs the baseline methods when trained over noisy data." *Id*. "Noisy data" is the phrase the paper used to refer to normal data supplemented with abnormal data. *Id.* at 5.

69. A paper by Professor Stolfo and others from 1999 similarly explains that a mixture of normal and abnormal data can be used to construct the model of normal behavior. Ex. O to Snyder Decl., Wenke Lee, Salvatore J. Stolfo & Kui W. Mok, *Mining in a Data-Flow Environment: Experience in Network Intrusion Detection*, Proceedings of the fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (1999). The paper first constructs a model based on "normal patterns." *Id*. at 4. The paper then supplements this data with anomalous data (data on "intrusion only patterns") to construct additional features of the model: "Each of the intrusion only patterns . . . is used for constructing additional features into the connection records . . . ." *Id*. at 5–6.

70. Another paper cited in the specification by Professor Stolfo from 1997 uses normal and abnormal data to construct "normal patterns (classifiers)": "We applied a machine learning approach to learn normal and abnormal patterns of program behavior . . . . The resultant

normal patterns (classifiers) are shown to be able to accurately detect anomalous intrusions." Ex. P to Snyder Decl., Wenke Lee, Salvatore J. Stolfo &Philip K. Chan, *Learning Patterns from Unix Process Execution Traces for Intrusion Detection*, AAAI Workshop on AI Approaches to Fraud Detection and Risk Management (1997), at 6. It thus was generally recognized in the field at the time of the patents that anomaly detection and constructing models of normal activity does not exclude the consideration of abnormal data.

71. It would have been readily apparent to persons of skill in the art that in the '084/'306 patent, a model of normal computer system usage must involve the use of data on normal system activity, but it can include additional information as well.

72. The concept of a "probabilistic model of normal computer system usage" is also readily understood. Probabilistic models are models that employ probability. This is made explicit in the patent. Ex. E to Snyder Decl., '084 patent at 3:43–46 ("The step of generating a probabilistic model of normal computer system usage may comprise determining a likelihood of observing a feature in the records of processes that access the Windows registry."). Probability is likewise easily understood: a probability is the likelihood that an event will occur or a condition will be present. Ex. Q to Snyder Decl., Pocket Dictionary on Statistics 210 (2002) ("probability-A numerical measure of the likelihood that an event will occur.").

VIII. '115/'322 Patents

73. United States Patent Nos. 8,074,115 and 8,601,322 share a common parent application filed in October 25, 2006, based on a provisional application filed on October 25, 2005. I understand that I am to provide my opinions on the understanding of concepts by a person of ordinary skill as of the filing date of the application. No opinion I give changes if the date selected October 2006 or October 2005.

A. Background on the ’115/’322 Patents

74. To understand the ’115/’322 patents, it is important to have some context on the issues the inventors were facing when they did their work.

75. This family of patents relates to techniques to determine whether the behavior of a program is indicative of maliciousness. This is an important area of research for at least two reasons.

76. First, the danger posed by an unknown program may not be apparent by simply examining the program before it runs. Some malicious characteristics may not manifest themselves until the program is executed. Therefore, risk can only be fully assessed by studying the program while it is running.

77. Second, there can be a significant performance penalty from running the entire program in some kind of safe environment (where it cannot infect other parts of the computer or the network connected to the computer). The process of running and studying the entire program in a safe environment will take time. A good analogy is what happens when a pathogen is sent for study to biohazard laboratory at a place like the Centers for Disease Control. This process is very safe, but the patient infected with the pathogen must wait a long time for the diagnosis.

78. Claim 1 of the ’115 patent is exemplary:

1. A method for detecting anomalous program executions, comprising:

> executing at least a part of a program in an emulator;
>
> comparing a function call made in the emulator to a model of function calls for the at least a part of the program;
>
> identifying the function call as anomalous based on the comparison; and

> upon identifying the anomalous function call, notifying an application community that includes a plurality of computers of the anomalous function call.

In this design, the system detects anomalous activity by using an emulator that monitors and selectively executes "at least a part of a program." The monitored activity inside the emulator is then compared to a model. If an anomaly is detected, an application community of machines running the same program is notified. I discuss these core concepts in the following sections.

B. The Use of an Emulator

79. The patent teaches the use of an emulator component that can study a part or all of a program:

> In various embodiments, using PAD [Probablistic Anomaly Detection] to model program stack information, such stack information may be extracted using, for example, Selective Transactional EMulation (STEM), which is described below and which [***selective execution***] ***permits the selective execution of certain parts, or all, of a program inside an instruction-level emulator***, using the Valgrind emulator, by modifying a program's binary or source code to include indicators of what functions calls are being made (and any other suitable related information), or using any other suitable technique. In this manner, it [***monitoring***] ***is possible to determine dynamically (and transparently to the monitored program) the necessary information such as stack frames, function-call arguments, etc***. Ex. G to Snyder Decl., '115 patent at 3:28–40 (emphasis added).

The purpose of this emulator is to allow the security system to monitor and selectively execute particular parts (or all) of a program before it can harm the system. The program can be inspected in a safe way without allowing it unfettered access to the computing device.

80. Although the specification discusses a particular emulator called STEM as a preferred embodiment, the specification makes equally clear that it is not bound by any specific method of emulation. The specification mentions various kinds of emulators that are within the scope of the patents. The patent states how the particular examples of emulators described in the passage quoted above can be used as well as "any other suitable technique." *Id.* at 3:28–40.

81. The specification repeatedly references certain characteristics that define the emulator of the '115/'322 patents. There is a core concept of monitoring a program. For example, the specification states that "Figure 1 is a schematic diagram of an illustrative system suitable for implementation of an application that ***monitors*** other applications and protects these applications against faults. . . ." *Id.* at 2:43–46 (emphasis added). The specification also states that "The use of an emulator allows the system to ***detect and/or monitor*** a wide array of software failures . . . ." *Id.* at 14:16–19 (emphasis added); *see also id.* at 13:54–61. In a preferred embodiment, the monitoring can be achieved by actually inserting special code into the program so that the program can be monitored as portions of it are being run: "modifying a program's binary or source code to include indicators of what function calls are being made . . . ." *Id.* at 3:34–35.

82. Another core concept is the selective execution of all or a part of a program. For example, the specification introduces the emulator by stating how the emulator "***permits the selective execution of certain parts, or all, of a program…***" *Id.* at 3:28–40 (emphasis added). The specification states: "… the instruction-level emulator can be ***selectively invoked*** for segments of the application's code, thereby allowing the system to mix emulated and non-emulated code within the same code execution." *Id.* at 13:51–54 (emphasis added). The specification consistently emphasizes that the emulator only needs to selectively execute a portion of the program's code:

> Isolating a portion of the application's code and using the emulator on the portion allows the system to reduce and/or minimize the performance impact on the immunized application. However, while this embodiment isolates a portion or a slice of the application's code, the entire application may also be emulated. *Id*. at 13:14–19.

83. A figure corresponding to a preferred embodiment depicts inserting code into an application to monitor and selectively execute only a portion of the application:

500

```
void foo() {
    int a = 1;
    emulate_init();
    emulate_begin(p_args);
    a++;
    emulate_end();
    emulate_term();
    printf("a = %d\n", a);
}
```

FIG. 5

Id. at Fig. 5, 2:57–59 ("Fig. 5 shows an illustrative example of emulated code integrated into the code of an existing application in accordance with some embodiments.").

84. Being able to monitor and selectively execute portions of a program is important for two reasons. First, it is efficient, since the computer need not perform large-scale emulation of an entire program. Instead, only the portions of the code that are particularly dangerous need be monitored. *Id.* at 9:8–14.

85. Second, the ability to monitor and selectively execute only parts of a program flows into a second important concept in the patent, notifying an "application community."

C. Application Community

86. The specification of the '115/'322 patents explains what an application community is:

> According to various embodiments, models are shared among many members of a community running the same application (referred to as an "application community"). In particular, some embodiments can share models with each other and/or update each other's models such that the learning of anomaly detection models is relatively quick. For example, instead of running a particular application for days at a single site, according to various embodiments, thousands

> of replicated applications can be run for a short period of time (e.g., one hour), and the models created based on the distributed data can be shared. While only a portion of each application instance may be monitored, for example, the entire software body can be monitored across the entire community. This can enable the rapid acquisition of statistics, and relatively fast learning of an application profile by sharing, for example, aggregate information (rather than the actual raw data used to construct the model). *Id.* at 6:31–47.

An application community, according to the specification, is members of a community running the same program or a selected portion of the program.

87. This passage also describes a strategy in which multiple different devices can each monitor and selectively execute a part of a program. These members of the community can be, for example, two "workstation[s] [or] server[s]." *Id.* at 16:54–60. For example, Figure 6 describes an embodiment in which multiple emulators on different computers ("devices") are assigned different parts of the program to study. What each computer does is "monitor[] the portion of the code[]" it has been assigned. *Id.* at Fig. 6, 2:59–63, 16:47–17:24.

D. Use of a Model to Detect Potential Danger

88. The patent teaches the comparison of function calls made by the monitored programs to models in order to detect whether there is a danger. *See, e.g.*, *id.* at 2:56–64. This concept of danger is reflected in the term "anomalous," which in this context means a divergence from normal that may correspond to an attack. At the time of the patents and currently this would be an accepted meaning. *See, e.g.,* Ex. I to Snyder Decl., McGraw-Hill Dictionary of Scientific and Technical Terms 108 (6th ed. 2002) ("anomalous [SCI TECH] Deviating from normal; irregular.").

89. The preferred embodiment discusses models of normal behavior that are used to detect anomalies. Ex. G to Snyder Decl., '115 patent at 4:9–14. One important way of determining what is normal is to look at how normal processes act. Although this is an important piece of the puzzle that needs to be present in a model of normal activity, it is not the only piece

of the puzzle. On the contrary, normal data (data point A) may be combined with malicious data (data point B) in order to create a model of what is normal. With this combination of data, one can conclude that normal looks like data point A, and normal doesn't look like data point B.

90. A model of normal activity does not mean that the data used to create that model is exclusively normal. Specifying that a model is one of normal behavior and relies on normal data is important because it distinguishes analysis that relies ***solely*** on malicious data. But these observations do not mean that ***only*** normal data must be used. In many situations, to focus solely on normal data would be to blind your security system to a whole host of important information on how malicious programs tend to act. There may be situations in which it is makes sense to only rely on 100% normal data (for example, when performing a proof of principle on a design), but there is nothing in the patent, or in the field, that mandates the exclusive reliance on normal data when constructing a model.

E. Once an Anomaly Is Detected, Certain Embodiments in the Patents Allow for Additional Functions to Occur

91. The core invention of the '115 patent as depicted in, for example, claim 1, involves the use of an emulator that can monitor and selectively execute a program, and then comparing function calls made in the emulator to a model in order to detect anomalies:[5]

> 1. A method for detecting anomalous program executions, comprising:
>
> executing at least a part of a program in an emulator;
>
> comparing a function call made in the emulator to a model of function calls for the at least a part of the program;
>
> identifying the function call as anomalous based on the comparison; and

[5] This is not intended to be a complete description of what is in the claim.

> upon identifying the anomalous function call, notifying an application community that includes a plurality of computers of the anomalous function call.

92. The patent teaches however, that a large amount of additional functions can potentially be added to the security system. For example, the patent states:

> In addition to monitoring for failures prior to executing instructions and reverting memory changes made by a particular function when a failure occurs (e.g., by having the emulator store memory modifications made during its execution), the emulator can also simulate an error return from the function. For example, some embodiments may generate a map between a set of errors that may occur during an application's execution and a limited set of errors that are explicitly handled by the application's code (sometimes referred to herein as "error virtualization"). As described below, the error virtualization features may be based on heuristics. However, any suitable approach for determining the return values for a function may be used. For example, aggressive source code analysis techniques to determine the return values that are appropriate for a function may be used. In another example, portions of code of specific functions can be marked as fail- safe and a specific value may be returned when an error return is forced (e.g., for code that checks user permissions). In yet another example, the error value returned for a function that has failed can be determined using information provided by a programmer, system administrator, or any other suitable user. *Id.* at 15:12–32.

93. In this optional design, ***after*** the anomaly is detected, a strategy known as "error virtualization" can be deployed to allow a program to continue executing when it would ordinarily crash or abort. The error virtualization strategy adds a separate functionality, simulating, that is not necessarily included in every emulator. In the context of the patents, emulation focuses on monitoring and selectively executing the actual code. Simulation, however, can be based on estimates and approximations about how the code runs. The patent recognizes this distinction when it discusses the use of heuristics in the context of error virtualization. *Id.* at 16:4–16 ("The appropriate error value may be determined based at least in part on ***heuristics***.") (emphasis added).

94. Consider the following example of how error virtualization can work. An emulator is used to monitor various pieces of code in a program. For one of the pieces of code

that is monitored, an anomaly is detected. At this point, options are available. One option is to attempt to allow the program to continue execution. This process can occur by using "error virtualization," wherein the system can use a simulation to understand what impact the anomalous code caused on the system and allow the system to return to its state before the error occurred:

> Using error virtualization, when an exception occurs during the emulation or if the system detects that a fault has occurred, ***the system may return the program state to its original settings*** and force an error return from the currently executing function. To determine the appropriate error value, the system analyzes the declared type of function. In some embodiments, the system may analyze the declared type of function using, for example, a TXL script. Generally, TXL is a hybrid function and rule-based language that may be used for performing source-to-source transformation and for rapidly prototyping new languages and language processors. Based on the declared type of function, the system determines the appropriate error value and places it in the stack frame of the returning function. ***The appropriate error value may be determined based at least in part on heuristics.*** For example, if the return type is an int, a value of**-1** is returned. If the return type is an unsigned int, the system returns a 0. If the function returns a pointer, the system determines whether the returned pointer is further dereferenced by the parent function. If the returned pointed is further dereferenced, the system expands the scope of the emulation to include the parent function. In another example, the return error code may be determined using information embedded in the source code of the application, or through additional information provided to the system by the application programmer, system administrator or third party. *Id.* at 15:57–16:16 (emphasis added).

95. Error virtualization is the context in which the specification talks about simulation. It is an advanced, optional feature discussed in connection with "error virtualization." It is not a defining characteristic of every type of emulator covered by the '115/'322 patents.

96. I declare under penalty of perjury that the foregoing is true and correct to the best of my knowledge.

Executed August 15 2014, in Richmond, Virginia.

Professor Douglas C. Szajda

Appendix A

I have been asked to apply the following standards and instructions in preparing my opinions.

Claim Construction Does Not Import Limitations from the Specification into Claim Terms

I understand that patent claims are generally interpreted in light of the "intrinsic" evidence of record, including the patent claims, the specification, the drawings, and the "prosecution history".[6] I understand that the starting point for proper claim construction is the words of the claim itself. I also understand that limitations from a single embodiment will not be read into a claim unless the patentee has demonstrated a clear intention to limit the claim scope using words or expressions of manifest exclusion or restriction.

Each Claim Is Presumed to Be Different in Scope

I understand that an independent claim is one that stands on its own without cross-reference to another claim in the patent. A dependent claim refers back to, and further limits, another claim in the patent.

The Level of Skill of a Person of Ordinary Skill in the Art

I understand that the words of a claim are generally given their ordinary and customary meaning, as understood by a person of ordinary skill in the art at the time of the invention. I understand that certain factors are considered when determining the level of skill of a person of ordinary skill in the art, including the inventor's level of skill. I understand that common factors include: (1) the educational level of the inventor; (2) type of problems encountered in the art;

[6] I understand that the "prosecution history" is the record of exchanges between the Patent Office and Columbia that led to the patents.

(3) prior art solutions to those problems; (4) rapidity with which innovations are made; (5) sophistication of the technology; and (6) educational level of workers in the field.

Appendix B

List of Materials Considered

1. Michael Riley, Ben Elgin, Dune Lawrence & Carol Matlack, *Missed Alarms and 40 Million Stolen Credit Card Numbers: How Target Blew It*, Bloomberg Businessweek (Mar. 13, 2014), http://www.businessweek.com/articles/2014-03-13/target-missed-alarms-in-epic-hack-of-credit-card-data.

2. United States Patent No. 7,487,544 and its prosecution history, including its provisional application.

3. United States Patent No. 7,979,907 and its prosecution history, including its provisional application.

4. United States Patent No. 7,448,084 and its prosecution history, including its provisional application.

5. United States Patent No. 7,913,306 and its prosecution history, including its provisional application.

6. United States Patent No. 8,074,115 and its prosecution history, including its provisional application.

7. United States Patent No. 8,604,322 and its prosecution history, including its provisional application.

8. Excerpts from McGraw-Hill Dictionary of Scientific and Technical Terms (6th ed. 2002).

9. An excerpt from Lee Holmes, *Windows PowerShell Cookbook* (2010).

10. Microsoft Corporation, *Microsoft Portable Executable and Common Object File Format Specification Revision 6.0* (1999).

11. IBM, Subodh Soni, *Understanding Linux configuration files*, IBM developerWorks (December 1, 2001), http://www.ibm.com/developerworks/library/l-config/.

12. Red Hat, *Configuring OpenSSH*, Red Hat Enterprise Linux, https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/System_Administrators_Guide/s1-ssh-configuration.html.

13. Eleazar Eskin, *Anomaly detection over noisy data using learned probability distributions*, Proceedings of the International Conference on Machine Learning (2000).

14. Wenke Lee, Salvatore J. Stolfo & Kui W. Mok, *Mining in a data-flow environment: Experience in network intrusion detection*, Proceedings of the fifth ACM SIGKDD international Conference on Knowledge discovery and data mining (1999).

15. Wenke Lee, Salvatore J. Stolfo &Philip K. Chan, *Learning patterns from unix process execution traces for intrusion detection*, AAAI Workshop on AI Approaches to Fraud Detection and Risk Management (1997), at 6.

16. An excerpt from Pocket Dictionary on Statistics (2002).

17. Transcript of the deposition of Andrew Honig conducted on August 6, 2014.